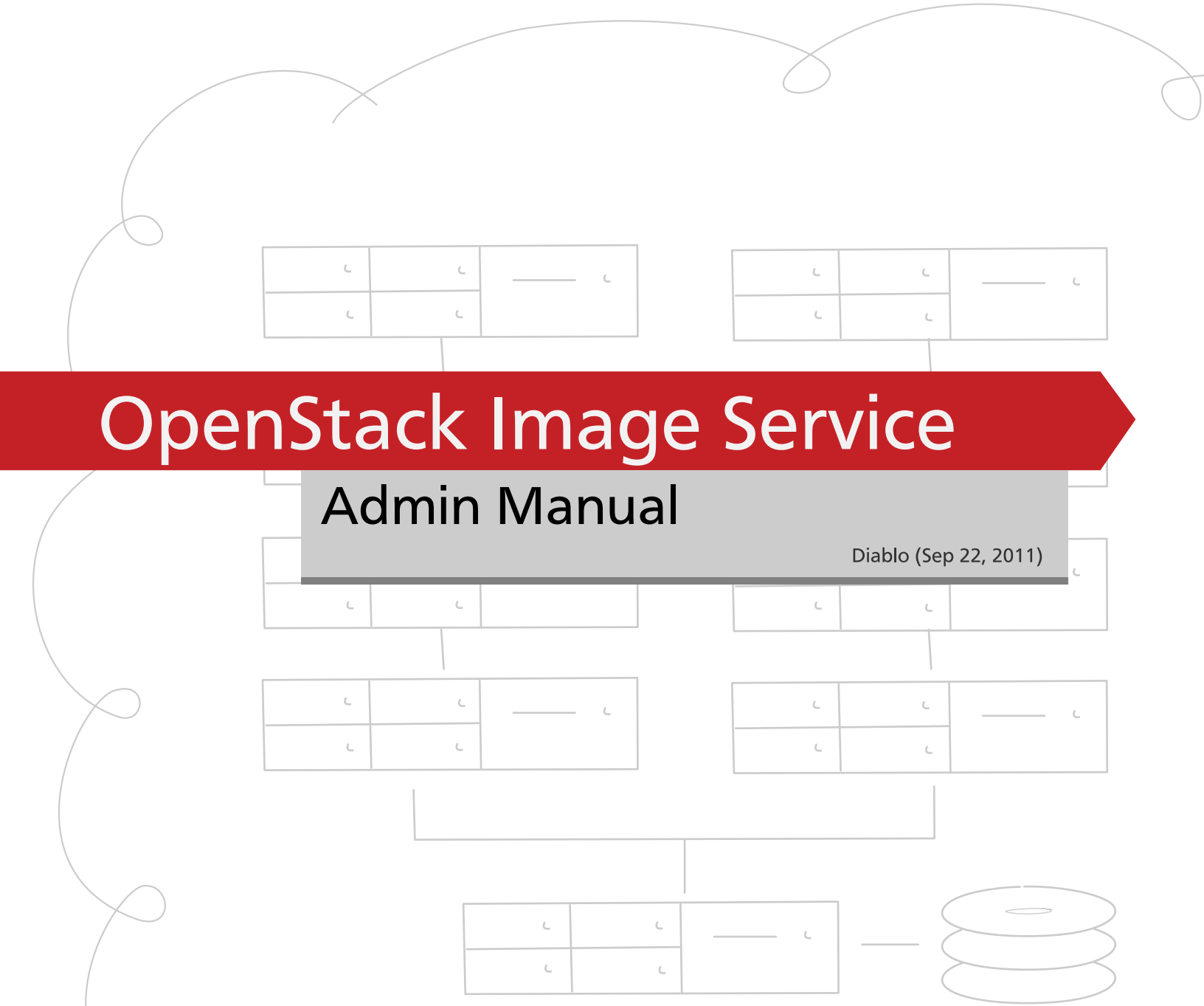

OpenStack Image Service

Admin Manual

Diablo (Sep 22, 2011)

openstack™

docs.openstack.org

OpenStack Image Service Admin Manual

Diablo (2011-09-22)
Copyright © 2010, 2011 OpenStack LLC All rights reserved.

OpenStack™ Image Service offers a service for discovering, registering, and retrieving virtual machine images. Code-named Glance, it has a RESTful API that allows querying of VM image metadata as well as retrieval of the actual image. This manual provides guidance for installing, managing, and understanding the software that runs OpenStack Image Service.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Table of Contents

1. Quick Guide to Getting Started with Glance 1
Overview of Glance Architecture 1
Glance API Server 1
Glance Registry Servers 2
2. Installing Glance 3
Installing from packages 3
Debian/Ubuntu 3
Installing from source tarballs 3
Installing from a Bazaar Branch 3
Debian/Ubuntu 3
3. Image Identifiers 5
4. Image Registries 6
Glance Registry API 6
API in Summary 6
Filtering Images Returned via GET /images and GET /images/detail 6
POST /images 7
Examples 7
5. Image Statuses 8
6. Disk and Container Formats 9
Disk Format 9
Container Format 10
7. Controlling Glance Servers 11
Starting a server 11
Manually starting the server 11
Using the glance-control program to start the server 13
Stopping a server 14
Restarting a server 14
8. Configuring Glance 15
Common Configuration Options in Glance 15
Configuring Logging in Glance 16
Logging Options Available Only in Configuration Files 16
Configuring Glance Storage Backends 17
Configuring the Filesystem Storage Backend 17
Configuring the Swift Storage Backend 17
Configuring the S3 Storage Backend 19
Configuring the Glance Registry 20
Configuring Notifications 20
9. Using the Glance CLI Tool 22
The help command 22
The add command 24
Important Information about Uploading Images 24
Store virtual machine image data and metadata 24
Register a virtual machine image in another location 25
The update command 26
The delete command 27
The index command 27
The details command 27
The show command 28

The `clear` command 29
The `image-members` Command 29
The `member-images` Command 29
The `member-add` Command 29
The `member-delete` Command 30
The `members-replace` Command 30
10. Using Glance Programmatically with Glance's Client 31
Requesting a List of Public VM Images 31
Requesting Detailed Metadata on Public VM Images 31
Filtering Images Returned via `get_images()` and `get_images_detailed()` 31
Sorting Images Returned via `get_images()` and `get_images_detailed()` 32
Requesting Detailed Metadata on a Specific Image 33
Retrieving a Virtual Machine Image 33
Adding a New Virtual Machine Image 34
Requesting Image Memberships 36
Requesting Member Images 36
Adding a Member To an Image 37
Removing a Member From an Image 37
Replacing a Membership List For an Image 37
11. Glance Authentication With Keystone 39
Sharing Images With Others 39

1. Quick Guide to Getting Started with Glance

Glance is a server that provides the following services:

- Ability to store and retrieve virtual machine images
- Ability to store and retrieve metadata about these virtual machine images
- FUTURE: Convert a virtual machine image from one format to another
- FUTURE: Help caching proxies such as Varnish or Squid cache machine images

Communication with Glance occurs via a REST-like HTTP interface.

However, Glance includes a Client class that makes working with Glance easy and straightforward.

As of the Cactus release, there are also command-line tools for interacting with Glance.

Overview of Glance Architecture

There are two main parts to Glance's architecture:

- Glance API server
- Glance Registry server(s)

Glance API Server

The API server is the main interface for Glance. It routes requests from clients to registries of image metadata and to its **backend stores**, which are the mechanisms by which Glance actually saves incoming virtual machine images.

The backend stores that Glance can work with are as follows:

- **Swift**

  Swift is the highly-available object storage project in OpenStack. More information can be found about Swift here.

- **Filesystem**

  The default backend that Glance uses to store virtual machine images is the filesystem backend. This simple backend writes image files to the local filesystem.

- **S3**

  This backend allows Glance to store virtual machine images in Amazon's S3 service.

- **HTTP**

Glance can read virtual machine images that are available via HTTP somewhere on the Internet. This store is **readonly**

Glance Registry Servers

Glance registry servers are servers that conform to the Glance Registry API. Glance ships with a reference implementation of a registry server that complies with this API (`glance-registry`).

For more details on Glance's architecture see the Architecture section. For more information on what a Glance registry server is, see the Registries section.

2. Installing Glance

Installing from packages

To install the latest version of Glance from the Launchpad Bazaar repositories, following the following instructions.

Debian/Ubuntu

1. Add the Glance PPA to your sources.lst:

   $> sudo add-apt-repository ppa:glance-core/trunk $> sudo apt-get update

2. Install Glance:

   $> sudo apt-get install glance

Installing from source tarballs

To install the latest version of Glance from the Launchpad Bazaar repositories, following the following instructions.

1. Grab the source tarball from Launchpad

2. Untar the source tarball:

   $> tar -xzf <FILE>

3. Change into the package directory and build/install:

   $> cd glance-<RELEASE> $> sudo python setup.py install

Installing from a Bazaar Branch

To install the latest version of Glance from the Launchpad Bazaar repositories, following the following instructions.

Debian/Ubuntu

1. Install Bazaar and build dependencies:

   $> sudo apt-get install bzr python-eventlet python-routes python-greenlet swift $> sudo apt-get install python-argparse python-sqlalchemy python-wsgiref python-pastedeploy

Note

If you want to build the Glance documentation locally, you will also want to install the python-sphinx package.

1. Branch Glance's trunk branch:

```
$> bzr branch lp:glance
```

2. Install Glance:

```
$> sudo python setup.py install
```

3. Image Identifiers

Images are uniquely identified by way of a URI that matches the following signature:

```
<Glance Server Location>/images/<ID>
```

where `<Glance Server Location>` is the resource location of the Glance service that knows about an image, and `<ID>` is the image's identifier that is *unique to that Glance server*.

4. Image Registries

Image metadata made available through Glance can be stored in image `registries`. Image registries are any web service that adheres to the Glance REST-like API for image metadata.

Glance comes with a server program `glance-registry` that acts as a reference implementation of a Glance Registry.

Please see the section about Controlling Servers for more information on starting up the Glance registry server that ships with Glance.

Glance Registry API

Any web service that publishes an API that conforms to the following REST-like API specification can be used by Glance as a registry.

API in Summary

The following is a brief description of the Glance API:

```
GET     /images         Return brief information about public images
GET     /images/detail  Return detailed information about public images
GET     /images/<ID>    Return metadata about an image in HTTP headers
POST    /images         Register metadata about a new image
PUT     /images/<ID>    Update metadata about an existing image
DELETE  /images/<ID>    Remove an image's metadata from the registry
```

Filtering Images Returned via `GET /images` and `GET /images/detail`

Both the `GET /images` and `GET /images/detail` requests take query parameters that serve to filter the returned list of images. The following list details these query parameters.

- `name=NAME`

  Filters images having a `name` attribute matching `NAME`.

- `container_format=FORMAT`

  Filters images having a `container_format` attribute matching `FORMAT`

- `disk_format=FORMAT`

  Filters images having a `disk_format` attribute matching `FORMAT`

- `status=STATUS`

  Filters images having a `status` attribute matching `STATUS`

- `size_min=BYTES`

Filters images having a `size` attribute greater than or equal to BYTES

- `size_max=BYTES`

  Filters images having a `size` attribute less than or equal to BYTES

These two resources also accept sort parameters:

- `sort_key=KEY`

  Results will be ordered by the specified image attribute KEY. Accepted values include `id`, `name`, `status`, `disk_format`, `container_format`, `size`, `created_at` (default) and `updated_at`.

- `sort_dir=DIR`

  Results will be sorted in the direction `DIR`. Accepted values are `asc` for ascending or `desc` (default) for descending.

POST /images

The body of the request will be a JSON-encoded set of data about the image to add to the registry. It will be in the following format:

```
{'image':
  {'id': <ID>|None,
   'name': <NAME>,
   'status': <STATUS>,
   'disk_format': <DISK_FORMAT>,
   'container_format': <CONTAINER_FORMAT>,
   'properties': [ ... ]
  }
}
```

The request shall validate the following conditions and return a `400 Bad request` when any of the conditions are not met:

- `status` must be non-empty, and must be one of **active**, **saving**, **queued**, or **killed**
- `disk_format` must be non-empty, and must be one of **ari**, **aki**, **ami**, **raw**, **iso**, **vhd**, **vdi**, **qcow2**, or **vmdk**
- `container_format` must be non-empty, and must be on of **ari**, **aki**, **ami**, **bare**, or **ovf**
- If `disk_format` *or* `container_format` is **ari**, **aki**, **ami**, then *both* `disk_format` and `container_format` must be the same.

Examples

..todo:: Complete examples for Glance registry API

5. Image Statuses

Images in Glance can be in one the following statuses:

- `queued`

  The image identifier has been reserved for an image in the Glance registry. No image data has been uploaded to Glance.

- `saving`

  Denotes that an image's raw data is currently being uploaded to Glance. When an image is registered with a call to `POST /images` and there is an `x-image-meta-location` header present, that image will never be in the `saving` status (as the image data is already available in some other location).

- `active`

  Denotes an image that is fully available in Glance.

- `killed`

  Denotes that an error occurred during the uploading of an image's data, and that the image is not readable.

- `deleted`

  Glance has retained the information about the image, but it is no longer available to use. An image in this state will be removed automatically at a later date.

- `pending_delete`

  This is similiar to `deleted`, however, Glance has not yet removed the image data. An image in this state is recoverable.

6. Disk and Container Formats

When adding an image to Glance, you are may specify what the virtual machine image's *disk format* and *container format* are.

This document explains exactly what these formats are.

Disk Format

The disk format of a virtual machine image is the format of the underlying disk image. Virtual appliance vendors have different formats for laying out the information contained in a virtual machine disk image.

You can set your image's container format to one of the following:

- **raw**

  This is an unstructured disk image format

- **vhd**

  This is the VHD disk format, a common disk format used by virtual machine monitors from VMWare, Xen, Microsoft, VirtualBox, and others

- **vmdk**

  Another common disk format supported by many common virtual machine monitors

- **vdi**

  A disk format supported by VirtualBox virtual machine monitor and the QEMU emulator

- **iso**

  An archive format for the data contents of an optical disc (e.g. CDROM).

- **qcow2**

  A disk format supported by the QEMU emulator that can expand dynamically and supports Copy on Write

- **aki**

  This indicates what is stored in Glance is an Amazon kernel image

- **ari**

  This indicates what is stored in Glance is an Amazon ramdisk image

- **ami**

  This indicates what is stored in Glance is an Amazon machine image

Container Format

The container format refers to whether the virtual machine image is in a file format that also contains metadata about the actual virtual machine.

There are two main types of container formats: OVF and Amazon's AMI. In addition, a virtual machine image may have no container format at all –basically, it's just a blob of unstructured data...

You can set your image's container format to one of the following:

- **ovf**

  This is the OVF container format

- **bare**

  This indicates there is no container or metadata envelope for the image

- **aki**

  This indicates what is stored in Glance is an Amazon kernel image

- **ari**

  This indicates what is stored in Glance is an Amazon ramdisk image

- **ami**

  This indicates what is stored in Glance is an Amazon machine image

7. Controlling Glance Servers

This section describes the ways to start, stop, and reload Glance's server programs.

Starting a server

There are two ways to start a Glance server (either the API server or the reference implementation registry server that ships with Glance):

- Manually calling the server program
- Using the `glance-control` server daemon wrapper program

We recommend using the second way.

Manually starting the server

The first is by directly calling the server program, passing in command-line options and a single argument for a `paste.deploy` configuration file to use when configuring the server application.

Note

Glance ships with an ``etc/`` directory that contains sample ``paste.deploy`` configuration files that you can copy to a standard configuration directory and adapt for your own uses. Specifically, bind_host must be set properly.

If you do `not` specify a configuration file on the command line, Glance will do its best to locate a configuration file in one of the following directories, stopping at the first config file it finds:

- `$CWD`
- `~/.glance`
- `~/`
- `/etc/glance`
- `/etc`

The filename that is searched for depends on the server application name. So, if you are starting up the API server, `glance-api.conf` is searched for, otherwise `glance-registry.conf`.

If no configuration file is found, you will see an error, like:

```
$> glance-api
ERROR: Unable to locate any configuration file. Cannot load application
 glance-api
```

Here is an example showing how you can manually start the `glance-api` server and `glance-registry` in a shell.:

```
$ sudo glance-api glance-api.conf --debug &
jsuh@mc-ats1:~$ 2011-04-13 14:50:12    DEBUG [glance-api]
 ****************************************************************************
2011-04-13 14:50:12    DEBUG [glance-api] Configuration options gathered from
 config file:
2011-04-13 14:50:12    DEBUG [glance-api] /home/jsuh/glance-api.conf
2011-04-13 14:50:12    DEBUG [glance-api] ================================
============
2011-04-13 14:50:12    DEBUG [glance-api] bind_host                     65.
114.169.29
2011-04-13 14:50:12    DEBUG [glance-api] bind_port                     9292
2011-04-13 14:50:12    DEBUG [glance-api] debug                         True
2011-04-13 14:50:12    DEBUG [glance-api] default_store                 file
2011-04-13 14:50:12    DEBUG [glance-api] filesystem_store_datadir      /
home/jsuh/images/
2011-04-13 14:50:12    DEBUG [glance-api] registry_host                 65.
114.169.29
2011-04-13 14:50:12    DEBUG [glance-api] registry_port                 9191
2011-04-13 14:50:12    DEBUG [glance-api] verbose                       False
2011-04-13 14:50:12    DEBUG [glance-api]
 ****************************************************************************
2011-04-13 14:50:12    DEBUG [routes.middleware] Initialized with method
 overriding = True, and path info altering = True
2011-04-13 14:50:12    DEBUG [eventlet.wsgi.server] (21354) wsgi starting up
 on http://65.114.169.29:9292/

$ sudo glance-registry glance-registry.conf &
jsuh@mc-ats1:~$ 2011-04-13 14:51:16     INFO [sqlalchemy.engine.base.Engine.
0x...feac] PRAGMA table_info("images")
2011-04-13 14:51:16     INFO [sqlalchemy.engine.base.Engine.0x...feac] ()
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Col
 ('cid', 'name', 'type', 'notnull', 'dflt_value', 'pk')
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (0,
 u'created_at', u'DATETIME', 1, None, 0)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (1,
 u'updated_at', u'DATETIME', 0, None, 0)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (2,
 u'deleted_at', u'DATETIME', 0, None, 0)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (3,
 u'deleted', u'BOOLEAN', 1, None, 0)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (4,
 u'id', u'INTEGER', 1, None, 1)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (5,
 u'name', u'VARCHAR(255)', 0, None, 0)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (6,
 u'disk_format', u'VARCHAR(20)', 0, None, 0)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (7,
 u'container_format', u'VARCHAR(20)', 0, None, 0)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (8,
 u'size', u'INTEGER', 0, None, 0)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (9,
 u'status', u'VARCHAR(30)', 1, None, 0)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row
 (10, u'is_public', u'BOOLEAN', 1, None, 0)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row
 (11, u'location', u'TEXT', 0, None, 0)
```

```
2011-04-13 14:51:16     INFO [sqlalchemy.engine.base.Engine.0x...feac] PRAGMA
 table_info("image_properties")
2011-04-13 14:51:16     INFO [sqlalchemy.engine.base.Engine.0x...feac] ()
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Col
 ('cid', 'name', 'type', 'notnull', 'dflt_value', 'pk')
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (0,
 u'created_at', u'DATETIME', 1, None, 0)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (1,
 u'updated_at', u'DATETIME', 0, None, 0)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (2,
 u'deleted_at', u'DATETIME', 0, None, 0)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (3,
 u'deleted', u'BOOLEAN', 1, None, 0)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (4,
 u'id', u'INTEGER', 1, None, 1)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (5,
 u'image_id', u'INTEGER', 1, None, 0)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (6,
 u'key', u'VARCHAR(255)', 1, None, 0)
2011-04-13 14:51:16    DEBUG [sqlalchemy.engine.base.Engine.0x...feac] Row (7,
 u'value', u'TEXT', 0, None, 0)

$ ps aux | grep glance
root     20009  0.7  0.1  12744  9148 pts/1    S    12:47   0:00 /usr/bin/
python /usr/bin/glance-api glance-api.conf --debug
root     20012  2.0  0.1  25188 13356 pts/1    S    12:47   0:00 /usr/bin/
python /usr/bin/glance-registry glance-registry.conf
jsuh     20017  0.0  0.0   3368   744 pts/1    S+   12:47   0:00 grep glance
```

Simply supply the configuration file as the first argument (the `etc/glance-api.conf` and `etc/glance-registry.conf` sample configuration files were used in the above example) and then any common options you want to use (`--debug` was used above to show some of the debugging output that the server shows when starting up. Call the server program with `--help` to see all available options you can specify on the command line.)

For more information on configuring the server via the `paste.deploy` configuration files, see the section entitled Configuring Glance servers.

Note that the server `daemonizes` itself by using the standard shell backgrounding indicator, &, in the previous example. For most use cases, we recommend using the `glance-control` server daemon wrapper for daemonizing. See below for more details on daemonization with `glance-control`.

Using the `glance-control` program to start the server

The second way to start up a Glance server is to use the `glance-control` program. `glance-control` is a wrapper script that allows the user to start, stop, restart, and reload the other Glance server programs in a fashion that is more conducive to automation and scripting.

Servers started via the `glance-control` program are always `daemonized`, meaning that the server program process runs in the background.

To start a Glance server with `glance-control`, simply call `glance-control` with a server and the word "start", followed by any command-line options you wish to provide. Start the server with `glance-control` in the following way:

```
$> sudo glance-control <SERVER> start [CONFPATH]
```

..note:

```
You must use the ``sudo`` program to run ``glance-control`` currently, as the
pid files for the server programs are written to /var/run/glance/
```

Here is an example that shows how to start the `glance-registry` server with the `glance-control` wrapper script.

::

$ sudo glance-control api start glance-api.conf Starting glance-api with /home/jsuh/glance.conf

$ sudo glance-control registry start glance-registry.conf Starting glance-registry with /home/jsuh/glance.conf

$ ps aux | grep glance root 20038 4.0 0.1 12728 9116 ? Ss 12:51 0:00 /usr/bin/python /usr/bin/glance-api /home/jsuh/glance-api.conf root 20039 6.0 0.1 25188 13356 ? Ss 12:51 0:00 /usr/bin/python /usr/bin/glance-registry /home/jsuh/glance-registry.conf jsuh 20042 0.0 0.0 3368 744 pts/1 S+ 12:51 0:00 grep glance

The same `paste.deploy` configuration files are used by `glance-control` to start the Glance server programs, and you can specify (as the example above shows) a configuration file when starting the server.

Stopping a server

If you started a Glance server manually and did not use the & backgrounding function, simply send a terminate signal to the server process by typing `Ctrl-C`

If you started the Glance server using the `glance-control` program, you can use the `glance-control` program to stop it. Simply do the following:

```
$> sudo glance-control <SERVER> stop
```

as this example shows:

```
$> sudo glance-control registry stop
Stopping glance-registry  pid: 17602  signal: 15
```

Restarting a server

You can restart a server with the `glance-control` program, as demonstrated here:

```
$> sudo glance-control registry restart etc/glance-registry.conf
Stopping glance-registry  pid: 17611  signal: 15
Starting glance-registry with /home/jpipes/repos/glance/trunk/etc/glance-
registry.conf
```

8. Configuring Glance

Glance has a number of options that you can use to configure the Glance API server, the Glance Registry server, and the various storage backends that Glance can use to store images.

Most configuration is done via configuration files, with the Glance API server and Glance Registry server using separate configuration files.

When starting up a Glance server, you can specify the configuration file to use (see the documentation on controller Glance servers). If you do **not** specify a configuration file, Glance will look in the following directories for a configuration file, in order:

- `~/.glance`
- `~/`
- `/etc/glance`
- `/etc`

The Glance API server configuration file should be named `glance-api.conf`. Similarly, the Glance Registry server configuration file should be named `glance-registry.conf`. If you installed Glance via your operating system's package management system, it is likely that you will have sample configuration files installed in `/etc/glance`.

In addition to this documentation page, you can check the `etc/glance-api.conf` and `etc/glance-registry.conf` sample configuration files distributed with Glance for example configuration files for each server application with detailed comments on what each options does.

Common Configuration Options in Glance

Glance has a few command-line options that are common to all Glance programs:

- `--verbose`

Optional. Default: `False`

Can be specified on the command line and in configuration files.

Turns on the INFO level in logging and prints more verbose command-line interface printouts.

- `--debug`

Optional. Default: `False`

Can be specified on the command line and in configuration files.

Turns on the DEBUG level in logging.

- `--config-file=PATH`

Optional. Default: `None`

Specified on the command line only.

Takes a path to a configuration file to use when running the program. If this CLI option is not specified, then we check to see if the first argument is a file. If it is, then we try to use that as the configuration file. If there is no file or there were no arguments, we search for a configuration file in the following order:

- `~/.glance`
- `~/`
- `/etc/glance`
- `/etc`

The filename that is searched for depends on the server application name. So, if you are starting up the API server, `glance-api.conf` is searched for, otherwise `glance-registry.conf`.

Configuring Logging in Glance

There are a number of configuration options in Glance that control how Glance servers log messages.

- `--log-config=PATH`

Optional. Default: `None`

Specified on the command line only.

Takes a path to a configuration file to use for configuring logging.

Logging Options Available Only in Configuration Files

You will want to place the different logging options in the **[DEFAULT]** section in your application configuration file. As an example, you might do the following for the API server, in a configuration file called `etc/glance-api.conf`:

```
[DEFAULT]
log_file = /var/log/glance/api.log
```

- `log_file`

The filepath of the file to use for logging messages from Glance's servers. If missing, the default is to output messages to `stdout`, so if you are running Glance servers in a daemon mode (using `glance-control`) you should make sure that the `log_file` option is set appropriately.

- `log_dir`

The filepath of the directory to use for log files. If not specified (the default) the `log_file` is used as an absolute filepath.

- `log_date_format`

The format string for timestamps in the log output.

Defaults to `%Y-%m-%d %H:%M:%S`. See the logging module documentation for more information on setting this format string.

Configuring Glance Storage Backends

There are a number of configuration options in Glance that control how Glance stores disk images. These configuration options are specified in the `glance-api.conf` config file in the section `[DEFAULT]`.

- `default_store=STORE`

Optional. Default: `file`

Can only be specified in configuration files.

Sets the storage backend to use by default when storing images in Glance. Available options for this option are (`file`, `swift`, or `s3`).

Configuring the Filesystem Storage Backend

- `filesystem_store_datadir=PATH`

Optional. Default: `/var/lib/glance/images/`

Can only be specified in configuration files.

`This option is specific to the filesystem storage backend.`

Sets the path where the filesystem storage backend write disk images. Note that the filesystem storage backend will attempt to create this directory if it does not exist. Ensure that the user that `glance-api` runs under has write permissions to this directory.

Configuring the Swift Storage Backend

- `swift_store_auth_address=URL`

Required when using the Swift storage backend.

Can only be specified in configuration files.

`This option is specific to the Swift storage backend.`

Sets the authentication URL supplied to Swift when making calls to its storage system. For more information about the Swift authentication system, please see the Swift auth documentation and the overview of Swift authentication.

- `swift_store_user=USER`

Required when using the Swift storage backend.

Can only be specified in configuration files.

`This option is specific to the Swift storage backend.`

Sets the user to authenticate against the `swift_store_auth_address` with.

- `swift_store_key=KEY`

Required when using the Swift storage backend.

Can only be specified in configuration files.

`This option is specific to the Swift storage backend.`

Sets the authentication key to authenticate against the `swift_store_auth_address` with for the user `swift_store_user`.

- `swift_store_container=CONTAINER`

Optional. Default: `glance`

Can only be specified in configuration files.

`This option is specific to the Swift storage backend.`

Sets the name of the container to use for Glance images in Swift.

- `swift_store_create_container_on_put`

Optional. Default: `False`

Can only be specified in configuration files.

`This option is specific to the Swift storage backend.`

If true, Glance will attempt to create the container `swift_store_container` if it does not exist.

- `swift_store_large_object_size=SIZE_IN_MB`

Optional. Default: `5120`

Can only be specified in configuration files.

`This option is specific to the Swift storage backend.`

What size, in MB, should Glance start chunking image files and do a large object manifest in Swift? By default, this is the maximum object size in Swift, which is 5GB

- `swift_store_large_object_chunk_size=SIZE_IN_MB`

Optional. Default: `200`

Can only be specified in configuration files.

`This option is specific to the Swift storage backend.`

When doing a large object manifest, what size, in MB, should Glance write chunks to Swift? This amount of data is written to a temporary disk buffer during the process of chunking the image file, and the default is 200MB

Configuring the S3 Storage Backend

- `s3_store_host=URL`

Required when using the S3 storage backend.

Can only be specified in configuration files.

`This option is specific to the S3 storage backend.`

Default: s3.amazonaws.com

Sets the main service URL supplied to S3 when making calls to its storage system. For more information about the S3 authentication system, please see the S3 documentation

- `s3_store_access_key=ACCESS_KEY`

Required when using the S3 storage backend.

Can only be specified in configuration files.

`This option is specific to the S3 storage backend.`

Sets the access key to authenticate against the `s3_store_host` with.

You should set this to your 20-character Amazon AWS access key.

- `s3_store_secret_key=SECRET_KEY`

Required when using the S3 storage backend.

Can only be specified in configuration files.

`This option is specific to the S3 storage backend.`

Sets the secret key to authenticate against the `s3_store_host` with for the access key `s3_store_access_key`.

You should set this to your 40-character Amazon AWS secret key.

- `s3_store_bucket=BUCKET`

Required when using the S3 storage backend.

Can only be specified in configuration files.

`This option is specific to the S3 storage backend.`

Sets the name of the bucket to use for Glance images in S3.

Note that the namespace for S3 buckets is **global**, and therefore you must use a name for the bucket that is unique. It is recommended that you use a combination of your AWS access key, **lowercased** with "glance".

For instance if your Amazon AWS access key is:

ABCDEFGHIJKLMNOPQRST

then make your bucket value be:

abcdefghijklmnopqrstglance

- s3_store_create_bucket_on_put

Optional. Default: False

Can only be specified in configuration files.

`This option is specific to the S3 storage backend.`

If true, Glance will attempt to create the bucket s3_store_bucket if it does not exist.

Configuring the Glance Registry

Glance ships with a default, reference implementation registry server. There are a number of configuration options in Glance that control how this registry server operates. These configuration options are specified in the glance-registry.conf config file in the section [DEFAULT].

- sql_connection=CONNECTION_STRING (--sql-connection when specified on command line)

Optional. Default: None

Can be specified in configuration files. Can also be specified on the command-line for the glance-manage program.

Sets the SQLAlchemy connection string to use when connecting to the registry database. Please see the documentation for SQLAlchemy connection strings online.

- sql_timeout=SECONDS on command line)

Optional. Default: 3600

Can only be specified in configuration files.

Sets the number of seconds after which SQLAlchemy should reconnect to the datastore if no activity has been made on the connection.

Configuring Notifications

Glance can optionally generate notifications to be logged or sent to a RabbitMQ queue. The configuration options are specified in the glance-api.conf config file in the section [DEFAULT].

- `notifier_strategy`

Optional. Default: `noop`

Sets the strategy used for notifications. Options are `logging`, `rabbit` and `noop`.

- `rabbit_host`

Optional. Default: `localhost`

Host to connect to when using `rabbit` strategy.

- `rabbit_port`

Optional. Default: `5672`

Port to connect to when using `rabbit` strategy.

- `rabbit_use_ssl`

Optional. Default: `false`

Boolean to use SSL for connecting when using `rabbit` strategy.

- `rabbit_userid`

Optional. Default: `guest`

Userid to use for connection when using `rabbit` strategy.

- `rabbit_password`

Optional. Default: `guest`

Password to use for connection when using `rabbit` strategy.

- `rabbit_virtual_host`

Optional. Default: `/`

Virtual host to use for connection when using `rabbit` strategy.

- `rabbit_notification_topic`

Optional. Default: `glance_notifications`

Topic to use for connection when using `rabbit` strategy.

9. Using the Glance CLI Tool

Glance ships with a command-line tool for querying and managing Glance It has a fairly simple but powerful interface of the form:

```
Usage: glance <command> [options] [args]
```

Where <command> is one of the following:

- `help`

  Show detailed help information about a specific command

- `add`

  Adds an image to Glance

- `update`

  Updates an image's stored metadata in Glance

- `delete`

  Deletes an image and its metadata from Glance

- `index`

  Lists brief information about *public* images that Glance knows about

- `details`

  Lists detailed information about *public* images that Glance knows about

- `show`

  Lists detailed information about a specific image

- `clear`

  Destroys all **public** images and their associated metadata

This document describes how to use the `glance` tool for each of the above commands.

The `help` command

Issuing the `help` command with a <COMMAND> argument shows detailed help about a specific command. Running `glance` without any arguments shows a brief help message, like so:

```
$> glance
Usage: glance <command> [options] [args]
```

```
Commands:

    help <command>  Output help for one of the commands below

    add             Adds a new image to Glance

    update          Updates an image's metadata in Glance

    delete          Deletes an image from Glance

    index           Return brief information about images in Glance

    details         Return detailed information about images in
                    Glance

    show            Show detailed information about an image in
                    Glance

    clear           Removes all images and metadata from Glance

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -v, --verbose         Print more verbose output
  -H ADDRESS, --host=ADDRESS
                        Address of Glance API host. Default: example.com
  -p PORT, --port=PORT  Port the Glance API host listens on. Default: 9292
  --limit=LIMIT         Page size to use while requesting image metadata
  --marker=MARKER       Image index after which to begin pagination
  --sort_key=KEY        Sort results by this image attribute.
  --sort_dir=[desc|asc]
                        Sort results in this direction.
  -f, --force           Prevent select actions from requesting user
                        confirmation
  --dry-run             Don't actually execute the command, just print output
                        showing what WOULD happen.
```

With a <COMMAND> argument, more information on the command is shown, like so:

```
$> glance help update

glance update [options] <ID> <field1=value1 field2=value2 ...>

Updates an image's metadata in Glance. Specify metadata fields as arguments.

All field/value pairs are converted into a mapping that is passed
to Glance that represents the metadata for an image.

Field names that can be specified:

name                A name for the image.
is_public           If specified, interpreted as a boolean value
                    and sets or unsets the image's availability to the public.
disk_format         Format of the disk image
container_format    Format of the container

All other field names are considered to be custom properties so be careful
to spell field names correctly. :)
```

The add command

The `add` command is used to do both of the following:

- Store virtual machine image data and metadata about that image in Glance
- Let Glance know about an existing virtual machine image that may be stored somewhere else

We cover both use cases below.

Important Information about Uploading Images

Before we go over the commands for adding an image to Glance, it is important to understand that Glance **does not currently inspect** the image files you add to it. In other words, **Glance only understands what you tell it, via attributes and custom properties**.

If the file extension of the file you upload to Glance ends in '.vhd', Glance **does not** know that the image you are uploading has a disk format of `vhd`. You have to **tell** Glance that the image you are uploading has a disk format by using the `disk_format=vhd` on the command line (see more below).

By the same token, Glance does not currently allow you to upload "multi-part" disk images at once. **The common operation of bundling a kernel image and ramdisk image into a machine image is not done automagically by Glance.**

Store virtual machine image data and metadata

When adding an actual virtual machine image to Glance, you use the `add` command. You will pass metadata about the VM image on the command line, and you will use a standard shell redirect to stream the image data file to `glance`.

Let's walk through a simple example. Suppose we have a virtual disk image stored on our local filesystem that we wish to "upload" to Glance. This image is stored on our local filesystem in `/tmp/images/myimage.iso`.

We'd also like to tell Glance that this image should be called "My Image", and that the image should be public – anyone should be able to fetch it.

Here is how we'd upload this image to Glance. Change example ip number to your server ip number.:

```
$> glance add name="My Image" is_public=true < /tmp/images/myimage.iso --host=
65.114.169.29
```

If Glance was able to successfully upload and store your VM image data and metadata attributes, you would see something like this:

```
$> glance add name="My Image" is_public=true < /tmp/images/myimage.iso --host=
65.114.169.29
```

```
Added new image with ID: 2
```

You can use the `--verbose` (or `-v`) command-line option to print some more information about the metadata that was saved with the image:

```
$> glance --verbose add name="My Image" is_public=true < /tmp/images/myimage.
iso --host=65.114.169.29
Added new image with ID: 4
Returned the following metadata for the new image:
               container_format => ovf
                     created_at => 2011-02-22T19:20:53.298556
                        deleted => False
                     deleted_at => None
                    disk_format => raw
                             id => 4
                      is_public => True
                       location => file:///tmp/images/4
                           name => My Image
                     properties => {}
                           size => 58520278
                         status => active
                     updated_at => None
Completed in 0.6141 sec.
```

If you are unsure about what will be added, you can use the `--dry-run` command-line option, which will simply show you what *would* have happened:

```
$> glance --dry-run add name="Foo" distro="Ubuntu" is_publi=True < /tmp/
images/myimage.iso --host=65.114.169.29
Dry run. We would have done the following:
Add new image with metadata:
               container_format => ovf
                    disk_format => raw
                      is_public => False
                           name => Foo
                     properties => {'is_publi': 'True', 'distro': 'Ubuntu'}
```

This is useful for detecting problems and for seeing what the default field values supplied by `glance` are. For instance, there was a typo in the command above (the `is_public` field was incorrectly spelled `is_publi` which resulted in the image having an `is_publi` custom property added to the image and the *real* `is_public` field value being `False` (the default) and not `True`...

Register a virtual machine image in another location

Sometimes, you already have stored the virtual machine image in some non-Glance location – perhaps even a location you have no write access to – and you want to tell Glance where this virtual machine image is located and some metadata about it. The `add` command can do this for you.

When registering an image in this way, the only difference is that you do not use a shell redirect to stream a virtual machine image file into Glance, but instead, you tell Glance where to find the existing virtual machine image by setting the `location` field. Below is an example of doing this.

Let's assume that there is a virtual machine image located at the URL `http://example.com/images/myimage.vhd`. We can register this image with Glance using the following:

```
$> glance --verbose add name="Some web image" disk_format=vhd
 container_format=ovf\
   location="http://example.com/images/myimage.vhd"
Added new image with ID: 1
Returned the following metadata for the new image:
               container_format => ovf
                     created_at => 2011-02-23T00:42:04.688890
                        deleted => False
                     deleted_at => None
                    disk_format => vhd
                             id => 1
                      is_public => True
                       location => http://example.com/images/myimage.vhd
                           name => Some web image
                     properties => {}
                           size => 0
                         status => active
                     updated_at => None
Completed in 0.0356 sec.
```

The `update` command

After uploading/adding a virtual machine image to Glance, it is not possible to modify the actual virtual machine image – images are read-only after all –however, it *is* possible to update any metadata about the image after you add it to Glance.

The `update` command allows you to update the metadata fields of a stored image. You use this command like so:

```
glance update <ID> [field1=value1 field2=value2 ...]
```

Let's say we have an image with identifier 5 that we wish to change the is_public attribute of the image from False to True. The following would accomplish this:

```
$> glance update 5 is_public=true --host=65.114.169.29
Updated image 5
```

Using the `--verbose` flag will show you all the updated data about the image:

```
$> glance --verbose update 5 is_public=true --host=65.114.169.29
Updated image 5
Updated image metadata for image 5:
URI: http://example.com/images/5
Id: 5
Public? Yes
Name: My Image
Size: 58520278
Location: file:///tmp/images/5
Disk format: raw
Container format: ovf
```

```
Completed in 0.0596 sec.
```

The `delete` command

You can delete an image by using the `delete` command, shown below:

```
$> glance --verbose delete 5 --host=65.114.169.29
Deleted image 5
```

The `index` command

The `index` command displays brief information about the *public* images available in Glance, as shown below:

```
$> glance index --host=65.114.169.29
ID                Name                           Disk Format          Container
 Format     Size
----------------- ------------------------------ --------------------
 -------------------- --------------
1                 Ubuntu 10.10                   vhd                  ovf
                   58520278
2                 Ubuntu 10.04                   ami                  ami
                   58520278
3                 Fedora 9                       vdi                  bare
                        3040
4                 Vanilla Linux 2.6.22           qcow2                bare
                          0
```

Image metadata such as 'name', 'disk_format', 'container_format' and 'status' may be used to filter the results of an index or details command. These commands also accept 'size_min' and 'size_max' as lower and upper bounds of the image metadata 'size.' Any unrecognized fields are handled as custom image properties.

The 'limit' and 'marker' options are used by the index and details commands to control pagination. The 'marker' indicates the last record that was seen by the user. The page of results returned will begin after the provided image ID. The 'limit' param indicates the page size. Each request to the api will be restricted to returning a maximum number of results. Without the 'force' option, the user will be prompted before each page of results is fetched from the API.

Results from index and details commands may be ordered using the 'sort_key' and 'sort_dir' options. Any image attribute may be used for 'sort_key', while only 'asc' or 'desc' are allowed for 'sort_dir'.

The `details` command

The `details` command displays detailed information about the *public* images available in Glance, as shown below:

```
$> glance details --host=65.114.169.29
================================================================================
==
```

```
URI: http://example.com/images/1
Id: 1
Public? Yes
Name: Ubuntu 10.10
Status: active
Size: 58520278
Location: file:///tmp/images/1
Disk format: vhd
Container format: ovf
Property 'distro_version': 10.10
Property 'distro': Ubuntu
================================================================================
==
URI: http://example.com/images/2
Id: 2
Public? Yes
Name: Ubuntu 10.04
Status: active
Size: 58520278
Location: file:///tmp/images/2
Disk format: ami
Container format: ami
Property 'distro_version': 10.04
Property 'distro': Ubuntu
================================================================================
==
URI: http://example.com/images/3
Id: 3
Public? Yes
Name: Fedora 9
Status: active
Size: 3040
Location: file:///tmp/images/3
Disk format: vdi
Container format: bare
Property 'distro_version': 9
Property 'distro': Fedora
================================================================================
==
URI: http://example.com/images/4
Id: 4
Public? Yes
Name: Vanilla Linux 2.6.22
Status: active
Size: 0
Location: http://example.com/images/vanilla.iso
Disk format: qcow2
Container format: bare
================================================================================
==
```

The show command

The `show` command displays detailed information about a specific image, specified with `<ID>`, as shown below:

```
$> glance show 3 --host=65.114.169.29
URI: http://example.com/images/3
```

```
Id: 3
Public? Yes
Name: Fedora 9
Status: active
Size: 3040
Location: file:///tmp/images/3
Disk format: vdi
Container format: bare
Property 'distro_version': 9
Property 'distro': Fedora
```

The `clear` command

The `clear` command is an administrative command that deletes **ALL** images and all image metadata. Passing the `--verbose` command will print brief information about all the images that were deleted, as shown below:

```
$> glance --verbose clear --host=65.114.169.29
Deleting image 1 "Some web image" ... done
Deleting image 2 "Some other web image" ... done
Completed in 0.0328 sec.
```

The `image-members` Command

The `image-members` command displays the list of members with which a specific image, specified with <ID>, is shared, as shown below:

```
$> glance image-members 3 --host=65.114.169.29
tenant1
tenant2 *

(*: Can share image)
```

The `member-images` Command

The `member-images` command displays the list of images which are shared with a specific member, specified with <MEMBER>, as shown below:

```
$> glance member-images tenant1 --host=65.114.169.29
1
2 *

(*: Can share image)
```

The `member-add` Command

The `member-add` command grants a member, specified with <MEMBER>, access to a private image, specified with <ID>. The `--can-share` flag can be given to allow the member to share the image, as shown below:

```
$> glance member-add 1 tenant1 --host=65.114.169.29
$> glance member-add 1 tenant2 --can-share --host=65.114.169.29
```

The `member-delete` Command

The `member-delete` command revokes the access of a member, specified with <MEMBER>, to a private image, specified with <ID>, as shown below:

```
$> glance member-delete 1 tenant1
$> glance member-delete 1 tenant2
```

The `members-replace` Command

The `members-replace` command revokes all existing memberships on a private image, specified with <ID>, and replaces them with a membership for one member, specified with <MEMBER>. The `--can-share` flag can be given to allow the member to share the image, as shown below:

```
$> glance members-replace 1 tenant1 --can-share --host=65.114.169.29
```

The command is given in plural form to make it clear that all existing memberships are affected by the command.

10. Using Glance Programmatically with Glance's Client

While it is perfectly acceptable to issue HTTP requests directly to Glance via its RESTful API, sometimes it is better to be able to access and modify image resources via a client class that removes some of the complexity and tedium of dealing with raw HTTP requests.

Glance includes a client class for just this purpose. You can retrieve metadata about an image, change metadata about an image, remove images, and of course retrieve an image itself via this client class.

Below are some examples of using Glance's Client class. We assume that there is a Glance server running at the address `glance.example.com` on port `9292`.

Requesting a List of Public VM Images

We want to see a list of available virtual machine images that the Glance server knows about.

Using Glance's Client, we can do this using the following code

..code-block:: python

from glance.client import Client

c = Client("glance.example.com", 9292)

print c.get_images()

Requesting Detailed Metadata on Public VM Images

We want to see more detailed information on available virtual machine images that the Glance server knows about.

Using Glance's Client, we can do this using the following code

..code-block:: python

from glance.client import Client

c = Client("glance.example.com", 9292)

print c.get_images_detailed()

Filtering Images Returned via `get_images()` and `get_images_detailed()`

Both the `get_images()` and `get_images_detailed()` methods take query parameters that serve to filter the returned list of images.

When calling, simply pass an optional dictionary to the method containing the filters by which you wish to limit results, with the filter keys being one or more of the below:

- `name: NAME`

  Filters images having a `name` attribute matching `NAME`.

- `container_format: FORMAT`

  Filters images having a `container_format` attribute matching `FORMAT`

- `disk_format: FORMAT`

  Filters images having a `disk_format` attribute matching `FORMAT`

- `status: STATUS`

  Filters images having a `status` attribute matching `STATUS`

- `size_min: BYTES`

  Filters images having a `size` attribute greater than or equal to `BYTES`

- `size_max: BYTES`

  Filters images having a `size` attribute less than or equal to `BYTES`

Here's a quick example that will return all images less than or equal to 5G in size and in the `saving` status.

from glance.client import Client

c = Client("glance.example.com", 9292)

filters = {'status': 'saving', 'size_max': (5 * 1024 * 1024 * 1024)} print c.get_images_detailed(filters=filters)

Sorting Images Returned via `get_images()` and `get_images_detailed()`

Two parameters are available to sort the list of images returned by these methods.

- `sort_key: KEY`

  Images can be ordered by the image attribute `KEY`. Acceptable values: `id`, `name`, `status`, `container_format`, `disk_format`, `created_at` (default) and `updated_at`.

- `sort_dir: DIR`

  The direction of the sort may be defined by `DIR`. Accepted values: `asc` for ascending or `desc` (default) for descending.

The following example will return a list of images sorted alphabetically by name in ascending order.

..code-block:: python

from glance.client import Client

c = Client("glance.example.com", 9292)

print c.get_images(sort_key='name', sort_dir='asc')

Requesting Detailed Metadata on a Specific Image

We want to see detailed information for a specific virtual machine image that the Glance server knows about.

We have queried the Glance server for a list of public images and the data returned includes the `uri` field for each available image. This `uri` field value contains the exact location needed to get the metadata for a specific image.

Continuing the example from above, in order to get metadata about the first public image returned, we can use the following code

..code-block:: python

from glance.client import Client

c = Client("glance.example.com", 9292)

print c.get_image_meta("http://glance.example.com/images/1")

Retrieving a Virtual Machine Image

We want to retrieve that actual raw data for a specific virtual machine image that the Glance server knows about.

We have queried the Glance server for a list of public images and the data returned includes the `uri` field for each available image. This `uri` field value contains the exact location needed to get the metadata for a specific image.

Continuing the example from above, in order to get both the metadata about the first public image returned and its image data, we can use the following code

..code-block:: python

from glance.client import Client

c = Client("glance.example.com", 9292)

meta, image_file = c.get_image("http://glance.example.com/images/1")

print meta

f = open('some_local_file', 'wb') for chunk in image_<link xmlns:xlink="http://www.w3.org/1999/xlink" xlink:href="file:">file:</link> f.write(chunk) f.close()

Note

The return from Client.get_image() is a tuple of (`metadata`, `file`) where `metadata` is a mapping of metadata about the image and `file` is a generator that yields chunks of image data.

Adding a New Virtual Machine Image

We have created a new virtual machine image in some way (created a "golden image" or snapshotted/backed up an existing image) and we wish to do two things:

- Store the disk image data in Glance
- Store metadata about this image in Glance

We can do the above two activities in a single call to the Glance client. Assuming, like in the examples above, that a Glance API server is running at `glance.example.com`, we issue a call to `glance.client.Client.add_image`.

The method signature is as follows:

```
glance.client.Client.add_image(image_meta, image_data=None)
```

The `image_meta` argument is a mapping containing various image metadata. The `image_data` argument is the disk image data and is an optional argument.

The list of metadata that `image_meta` can contain are listed below.

- `name`

  This key/value is required. Its value should be the name of the image.

  Note that the name of an image *is not unique to a Glance node*. It would be an unrealistic expectation of users to know all the unique names of all other user's images.

- `id`

  This key/value is optional.

  When present, Glance will use the supplied identifier for the image. If the identifier already exists in that Glance node, then a `glance.common.exception.Duplicate` will be raised.

  When this key/value is *not* present, Glance will generate an identifier for the image and return this identifier in the response (see below)

- `store`

This key/value is optional. Valid values are one of `file`, `s3` or `swift`

When present, Glance will attempt to store the disk image data in the backing store indicated by the value. If the Glance node does not support the backing store, Glance will raise a `glance.common.exception.BadRequest`

When not present, Glance will store the disk image data in the backing store that is marked default. See the configuration option `default_store` for more information.

- `type`

  This key/values is required. Valid values are one of `kernel`, `machine`, `raw`, or `ramdisk`.

- `size`

  This key/value is optional.

  When present, Glance assumes that the expected size of the request body will be the value. If the length in bytes of the request body *does not match* the value, Glance will raise a `glance.common.exception.BadRequest`

  When not present, Glance will calculate the image's size based on the size of the request body.

- `is_public`

  This key/value is optional.

  When present, Glance converts the value to a boolean value, so "on, 1, true" are all true values. When true, the image is marked as a public image, meaning that any user may view its metadata and may read the disk image from Glance.

  When not present, the image is assumed to be *not public* and specific to a user.

- `properties`

  This key/value is optional.

  When present, the value is assumed to be a mapping of free-form key/value attributes to store with the image.

  For example, if the following is the value of the `properties` key in the `image_meta` argument:

```
{'distro': 'Ubuntu 10.10'}
```

  Then a key/value pair of "distro"/"Ubuntu 10.10" will be stored with the image in Glance.

  There is no limit on the number of free-form key/value attributes that can be attached to the image with `properties`. However, keep in mind that there is a 8K limit on the size of all HTTP headers sent in a request and this number will effectively limit the number of image properties.

If the `image_data` argument is omitted, Glance will add the `image_meta` mapping to its registries and return the newly-registered image metadata, including the new image's identifier. The `status` of the image will be set to the value `queued`.

As a complete example, the following code would add a new machine image to Glance

```
from glance.client import Client

c = Client("glance.example.com", 9292)

meta = {'name': 'Ubuntu 10.10 5G',
        'type': 'machine', 'is_public': True, 'properties': {'distro': 'Ubuntu 10.10'}}

new_meta = c.add_image(meta, open('/path/to/image.tar.gz'))

print 'Stored image. Got identifier: %s' % new_meta['id']
```

Requesting Image Memberships

We want to see a list of the other system tenants that may access a given virtual machine image that the Glance server knows about.

Continuing from the example above, in order to get the memberships for the image with ID 1, we can use the following code

```
from glance.client import Client

c = Client("glance.example.com", 9292)

members = c.get_image_members(1)
```

Note

The return from Client.get_image_members() is a list of dictionaries. Each dictionary has a `member_id` key, mapping to the tenant the image is shared with, and a `can_share` key, mapping to a boolean value that identifies whether the member can further share the image.

Requesting Member Images

We want to see a list of the virtual machine images a given system tenant may access.

Continuing from the example above, in order to get the images shared with 'tenant1', we can use the following code

```
from glance.client import Client

c = Client("glance.example.com", 9292)

images = c.get_member_images('tenant1')
```

Note

The return from Client.get_member_images() is a list of dictionaries. Each dictionary has an `image_id` key, mapping to an image shared with the member, and a `can_share` key, mapping to a boolean value that identifies whether the member can further share the image.

Adding a Member To an Image

We want to authorize a tenant to access a private image.

Continuing from the example above, in order to share the image with ID 1 with 'tenant1', and to allow 'tenant2' to not only access the image but to also share it with other tenants, we can use the following code

from glance.client import Client

c = Client("glance.example.com", 9292)

c.add_member(1, 'tenant1') c.add_member(1, 'tenant2', True)

..note:

```
The Client.add_member() function takes one optional argument, the `can_share`
value.  If one is not provided and the membership already exists, its current
`can_share` setting is left alone.  If the membership does not already exist,
then the `can_share` setting will default to `False`, and the membership will
be created.  In all other cases, existing memberships will be modified to use
the specified `can_share` setting, and new memberships will be created with
it.  The return value of Client.add_member() is not significant.
```

Removing a Member From an Image

We want to revoke a tenant's authorization to access a private image.

Continuing from the example above, in order to revoke the access of 'tenant1' to the image with ID 1, we can use the following code

from glance.client import Client

c = Client("glance.example.com", 9292)

c.delete_member(1, 'tenant1')

..note:

```
The return value of Client.delete_member() is not significant.
```

Replacing a Membership List For an Image

All existing image memberships may be revoked and replaced in a single operation.

Continuing from the example above, in order to replace the membership list of the image with ID 1 with two entries–the first allowing 'tenant1' to access the image, and the second allowing 'tenant2' to access and further share the image, we can use the following code

```
from glance.client import Client

c = Client("glance.example.com", 9292)

c.replace_members(1,
{'member_id': 'tenant1',
'can_share': False},
                  {'member_id': 'tenant2', 'can_share': True})
```

Note

The first argument to Client.replace_members() is the opaque identifier of the image; the remaining arguments are dictionaries with the keys `member_id` (mapping to a tenant name) and `can_share`. Note that `can_share` may be omitted, in which case any existing membership for the specified member will be preserved through the replace operation.

The return value of Client.replace_members() is not significant.

11. Glance Authentication With Keystone

Glance may optionally be integrated with Keystone. Setting this up is relatively straightforward: the Keystone distribution includes the requisite middleware and examples of appropriately modified `glance-api.conf` and `glance-registry.conf` configuration files in the `examples/paste` directory. Once you have installed Keystone and edited your configuration files, newly created images will have their `owner` attribute set to the tenant of the authenticated users, and the `is_public` attribute will cause access to those images for which it is `false` to be restricted to only the owner.

Note

The exception is those images for which `owner` is set to `null`, which may only be done by those users having the ``Admin`` role. These images may still be accessed by the public, but will not appear in the list of public images. This allows the Glance Registry owner to publish images for beta testing without allowing those images to show up in lists, potentially confusing users.

Sharing Images With Others

It is possible to allow a private image to be shared with one or more alternate tenants. This is done through image *memberships*, which are available via the `members` resource of images. (For more details, see :ref:`glanceapi`.) Essentially, a membership is an association between an image and a tenant which has permission to access that image. These membership associations may also have a `can_share` attribute, which, if set to `true`, delegates the authority to share an image to the named tenant.